Final Master Thesis

MASTER'S DEGREE IN AUTOMATIC CONTROL AND ROBOTICS

Design, modelling and control of a biped robot platform based on Poppy project.

MEMORY

Author: Joan Guasch Iglesias

Advisors: Dr. Cecilio Angulo, Dr. Manel Velasco

Date: October, 2016

Escuela Técnica Superior

de Ingeniería Industrial de Barcelona

Abstract

This thesis takes its basis on the Poppy Project, an open-source, 3D printed humanoid robotic platform, with the aim to become more affordable for the research community. So, a new design is required as well as providing all the knowledge to produce, control and experiment with a biped platform in real world. The new design is driven by the following objectives:

- Break the idea about the high cost of humanoid robot platforms.
- Reduce the complexity of 3D printed parts to be printed with any 3D printing technology.
- Get closer bipedal walking to any enthusiastic of robotics.
- Compatible with cutting edge software used nowadays.

In order to design and implement this new platform it was necessary to develop all the technological features of the robot (i.e. mechanics, assemble phase, electronics, software). Similarly to the original project, all the parts in this project are accessible following the open-source spirit and it becomes easy to be used due to the included libraries.

The new platform has been named *Poppy UPC*, a bipedal robot that keeps human proportions. The 6 degrees of freedom per leg provides a full control of the pose and orientation of each foot. To work comfortably, all the tools to control the robot has been developed using the ROS framework, which fits for bipedal walking algorithms.

Contents

List of Figures

List of Tables

Acronyms

ABS Acrylonitrile butadiene styrene

CAD Computer-Aided Design

CPU Central processing unit

DIN Deutsches Institut fÃijr Normung

DOF Degree of Freedom

FDM Fused Deposition Modeling

PCB Printed Circuit Board

PID ProportionalâĂŞintegralâĂŞderivative

PLA Poly(lactic acid)

PWM Pulse-width modulation

ROS Robot Operating System

SLA Stereolithography

SLS Selective laser sintering

STL STereoLithography

UART Universal asynchronous receiver/transmitter

UPC Universitat Politecnica de Catalunya

URDF Unified Robot Description Format

XML Extensible Markup Language

ZMP Zero Moment Point

Chapter 1

Introduction

1.1 Motivation

As an enthusiastic of robotics, availability of real robots to work with is always a dream. The cost of these platforms had been usually expensive, only affordable for large companies or research institutes. Moreover, since this field is still rapidly growing it is a hard possibility to find a job related to.

Spending working time on a robot helps to be aware of all the fields required as well as the consequences that can affect to the Society. Also, from an engineer point of view, allows to discover which fields fit better to yourself. Working on the correct field provides that each one offer his/her best. This is a reason why the main aim in this project was to design and implement something to help future students that start in robotics.

Sometimes inspiration comes from outside of our minds. This was the case for us. In 2012, the *Poppy project* [1] was released, showing to the community a humanoid platform with an interesting design. Due to the cost of some components, the idea to produce one by ourselves was discarded. However, Dimitris Zervas, a student in the Master's program, appeared one day, also interested in the Poppy project and knowing about the potential of this platform. So, both, decided to work together with the goal to generate a platform that could be used to study bipedal walking algorithms.

D. Zervas dedicates his master thesis [2] on producing more affordable actuators with the same or better properties that those from Dynamixel motors used in the Poppy project. So, to accomplish the general objective, this thesis continues this work by designing a complete platform taking as reference the original one. We are showing the same spirit: to provide an affordable platform for the robot community.

1.2 Objectives

The overall objective of this thesis is to build a humanoid robot platform suitable for studying bipedal walking algorithms. To archive this goal we based our project in a existing one, the Poppy Project, a Ph.D. Thesis developed by Dr. Matthieu Lapeyre. Both works share the same objective, but each one explores different paths. This thesis pursue the following purposes:

- Significant reduction of the cost of our platform with regards to the one in the Poppy project. Although the robot is open-source, the price of some elements makes this platform not sustainable for the usual robotic enthusiasts.
- Modification of the Dynamixel motors, an expensive actuator used like the basic one in the Poppy robot, for the modified servo made by Dimistris Zervas during his master thesis. These new motors provide better features with less than the half cost of the original actuators.
- Design of an adequate structure for the new actuators because a standard servo case not provides all the required elements for robotic applications.
- Due to the incorporation of these new actuators, redesign all the parts of the robot to make them compatible is mandatory. Eventually these changes will modify some of the original features. The new design also must simplify the production method, lowering the manufacturing process and assembling cost.
- Build the desired platform to proof the viability of the thesis and offer to the users a real platform to work on.
- Provide all the tools for novel users to comfortably develop their applications using the platform in a real environment or in a simulator. Hence, a set of libraries and examples must be elaborated to accomplish this task.

1.3 Scope

In the last design step, the proposed platform must be controlled to walk in a real scenario. However, this thesis focuses on deliver the programmable platform. A set of mechanics, electronics and software tasks are required, but control algorithms are left for future developments.

On the mechanical part, the platform must be robust an easy to be replicated. Accordingly, the thesis focuses on using at maximum standard hardware. The structure parts that are not standard can be produced using common 3D printing technologies.

Depending on the shapes or function of a part, some technologies are not able to perform its manufacturing, forcing the user to use the more expensive ones. For all that, the designed parts must be as easy as possible to be produce in a FDM printer, which offers the smallest operational cost.

In order to proof the viability of our proposal, a real platform is attached to this master thesis. Platform manufacturing, however, is not enough to prove its sustainability. The robotic platform must be able to perform movements. To accomplish this objective, an electronic solution has been developed to supply power to all the actuators and be able to communicate with them. The final platform is also offering a simple way to connect it to any king of CPU unit, from embedded computers to laptop o desktop computers.

To accomplish the objective to become an easy platform to work on for novel users, the master thesis requires to develop all the necessary files. These files include the firmware for the actuators, the physical description of the platform using the URDF standard, the configuration for creating a simulation of our platform, and the examples to start developing control algorithms. To reach all these specifications and allow to be used widely by the robotic community [3], several ROS packages were created, one for each type of task.

This master thesis does not cover the development of walking algorithms, the control examples being not tuned and serving only as a reference for future works. So this is the starting point for the users that will deal with this platform and those who we want to encourage.

1.4 State of the Art

1.4.1 Pioneer robots

The first robots [4] created were Elmer and Elsie in 1949, a pair of robots known as turtle robots created by the cybernetic pioneer W. Grey Walter. Although the turtle robot was entirely analogical, it was able to demonstrate complex behaviours. The robots were able to avoid obstacles and capable of finding their charging station when their battery power ran low.

In 1954 George Devol designed the first truly programmable robot and called it UNIMATE for “Universal Automation”. This robot was the first industrial robot which worked on a General Motors assembly line in 1961. But not was the only sector where robots started to work. In the 1960’s robots started to become part of the production line of several industries, even in a candy factory.

At the 1960’s, robotics started to spread around academic and industrial world. New

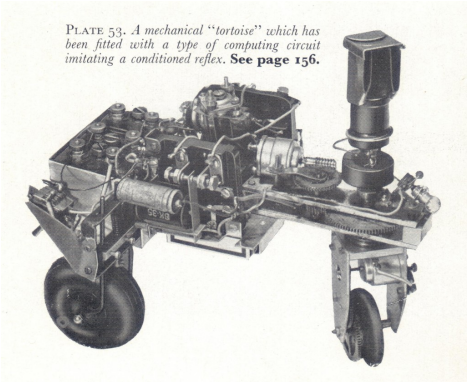

Figure 1.1: Elmer, the first robot ever created.

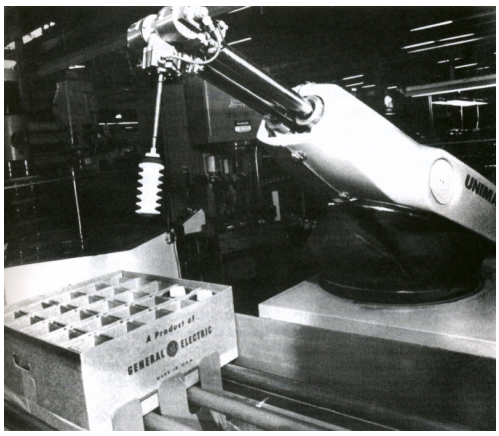

Figure 1.2: UNIMATE, the first industrial robot.

research institutes were created, developing in fields as artificial intelligence. A whole set of robotics platforms were created, in our case we focused in the walking platforms, specifically in biped robots.

In 1968 was created the Phony Pony, also knows as the Walking Horse and the California Horse. The first computer controlled walking machine was created by Mcgee and Frank at the University of South Carolina. The platform had a total of 8 DOF, 2 DOF per leg, one in the hip and other in the knee. The next years, more platforms with several legs where created like Hexapod (1977) or the Functionoid (1983). These latter platforms try to reproduce the walking behaviour of insects.

1.4.2 Early bipedal robots

In 1966, Professor Kato started studying not only a human-like robotic hand to be applicable to hand prostheses, but also a biped walking robot to analyse the human-walking mechanism. In 1967, an artificial biped walker, WL-1, was constructed on the basis of a human leg mechanism. In 1969, WL-3 was developed, which had an electro-hydraulic

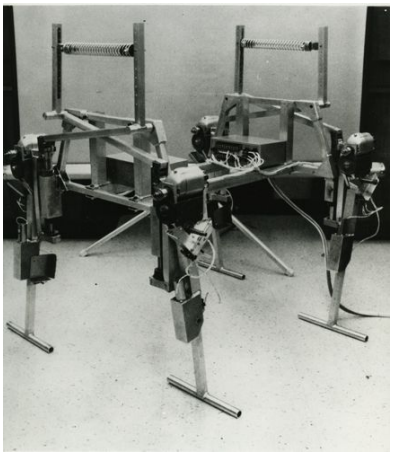

Figure 1.3: Phony Pony, the first walking robot.

servo-actuator and was controlled by using a master-slave method. In 1969, an anthropomorphic pneumatically activated pedipulator, WAP-1, was developed. Artificial muscles made of rubber were used for actuators. Planar biped locomotion was realized by teaching playback control. In 1970, WAP-2 was constructed, which had powerful pouch-type artificial muscles instead of actuators. It was controlled by automatic posture control based on pressure sensors implanted under the soles.

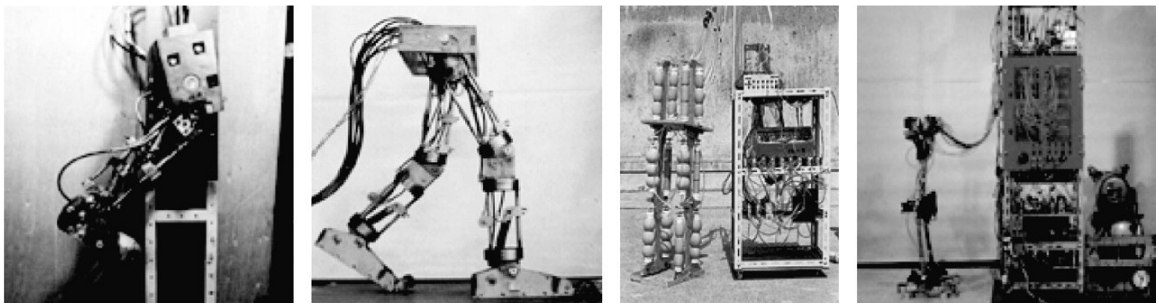

Figure 1.4: From left to right: WL-1, WL-3, WAP-1, WAP-2.

With the idea to design a robot able in the future to substitute the human labor power, the anthropomorphic intelligent robot WABOT [5] (WAseda roBOT) was developed. This is a platform aiming to finally develop a "personal robot" which resembles a person as much as possible. In 1970 The WABOT-1 was the first full-scale anthropomorphic robot developed in the world. It consisted of a limb-control system, a vision system and a conversation system.

1.4.3 Biped control

A new analysis of locomotion stability [6] was developed by M.Vukobratovic. M.Vukobratovic and his team were involved in the problems generated by functional rehabilitation. Around

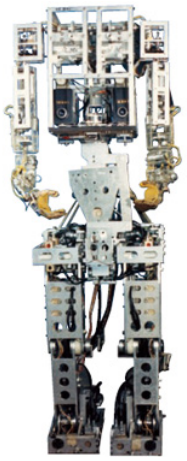

Figure 1.5: WABOT-1 robot.

1972 the concept of zero-moment point (ZMP) was exhibited. This was the first attempt to formalize the need for dynamical stability of legged robots. The idea was to use the dynamic wrench in order to extend a classical criterion of static balance: the center of mass should project inside the convex hull of contact points.

The ZMP concept was used in the next generation of walking robots, more robust and able to walk in irregular terrains. During the evolution of the ZMP, other platforms use other solutions, as the use of quasi-dynamic-walking, completed for the first time in the world by the model WL-9DR in 1979 using a 16bit microcomputer as its controller instead of a minicomputer, enabling versatile control.

In 1982, The model WL-10R [7] was constructed, in which the rotaries type servo-actuators were introduced and carbon fiber reinforced plastic was used in its structural parts. Finally the WL-10RD, developed in 1984, was the first ZMP-based robot, which successfully realized dynamic biped walking.

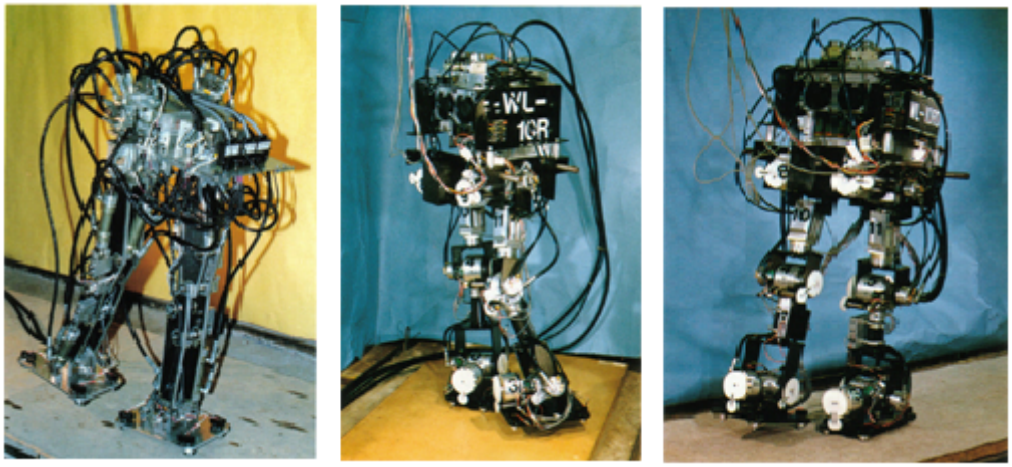

Figure 1.6: From left to right: WL-9DR, WL-10R, WL-10RD.

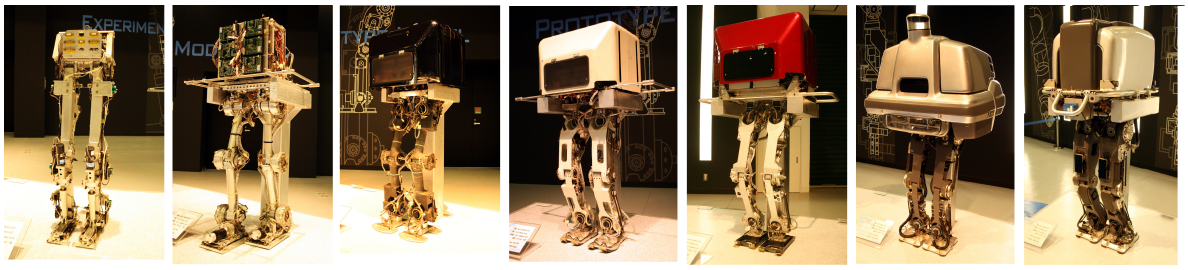

Figure 1.7: From left to right: E0, E1, E2, E3, E4, E5, E6.

In the 1990's another research in legged robots started by R.McGhee [8] with the aim of studying purely passive mechanical systems. This study evolve to the analysis of actuated robots from an energy point of view. This research generated two main approaches: the use of forward dynamics on one hand, and the use of the ZMP on the other hand.

The end of the millennium was a period of intense technological activities. Industrial companies showed to the world that building true humanoids was now possible. From 1986 until 1993, Honda developed a collection of successive humanoid robots called E-series.

1.4.4 Humanoid robots

The last two decades the diversity of humanoid robots becomes wider, having nowadays more than 30 different platforms. Starting in 2000 with the famous ASIMO created by Honda until the OceanOne, a humanoid diving robot exhibited this year.

These robots can be classified in human-size or reduced-size. As human-Size, the most popular are: ASIMO (2000), HRP-2 (2002), Johnnie (2003), RoboTurk (2006), REEM-A (2006), MAHRU (2006), REEM-B (2008), SURENA (2008), HRP-4C (2009), Kobian (2009), SURENA II (2010), SCHAFT (2013) and REEM-C (2013).

The reduced-size robots are perfect for bipedal walking studies because they can offer a real platform with a more affordable cost. Some of them has become commercial platforms and they can be considered as academic platforms or high-tech toys. The most popular are: HOAP-1 (2001), QRIO (2003), KHR-1 (2004), Nao (2006), iCub (2006), KT-X (2008), DARwIn-OP (2009), COMAN (2012), Poppy (2012) and Manav (2014).

One important fact is that the last two robots mentioned are 3D printed platforms. So this can be pointing to the new evolution of reduced-size humanoid robots. Moreover, both use expensive actuators regarding to the cost of the rest of the components. All of these facts are why we decided to develop a more affordable platform. Following the 3D printing technology as production method and modified servos as new actuators.

Chapter 2

Actuators

2.1 Modified servos

The motors that were modified are from Hitec and more specifically the model HS-7954SH. All the original electronics parts were removed and a new PCB board was designed in order to host a motor driver, a magnetic encoder (position sensor) and an arduino micro-controller (uC). The result is shown in Figure 2.1: the magnetic encoder is on a "daughter" board that is mounted underneath the "main" board that is shown.

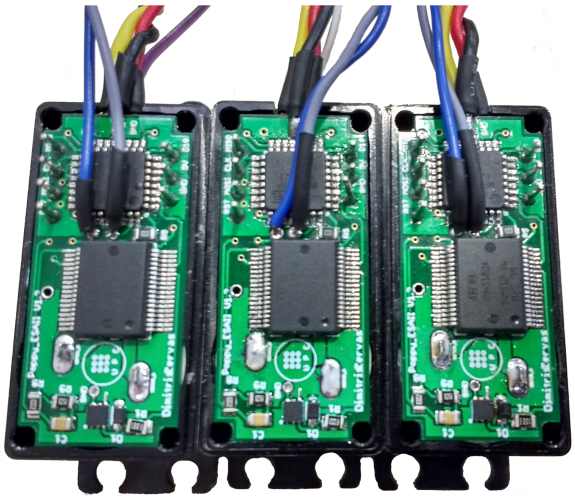

Figure 2.1: PCB board of modified servo

2.2 Specifications

Starting from the sensor, the modified servo is using the Austria Microsystems magnetic encoder, AS5145 with 12-bit resolution, which means 0.088 degrees of accuracy. About the driver, it uses a VNH5180A-E that can output up to 8A and provides under-voltage

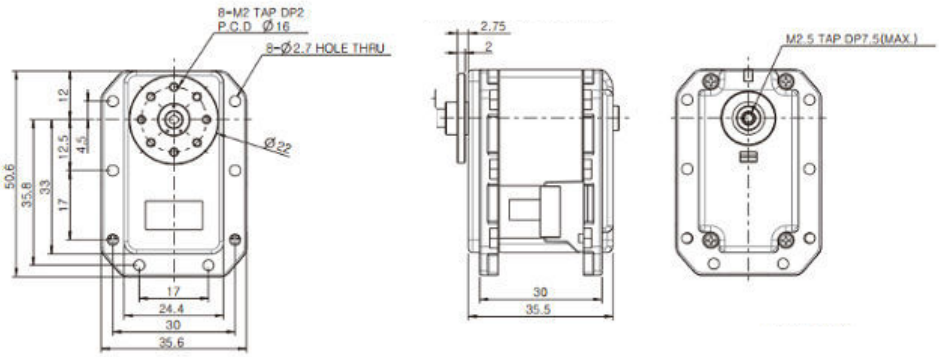

Figure 2.2: Dimensions and screw distribution of MX-28AT

shutdown, thermal shutdown, cross-conduction protection and the most important, a current sense output. To get a better idea about these parameters we provide a comparison between the Dynamixel motor MX-28AT, the one used in Poppy robot, and our modified servo in Table 6.2.

	Dynamixel MX-28AT	Hitect HS-7954SH
Operating voltage	12V	7.4V
Stall Torque	2.5 Nm	2.84 Nm
No-load speed	55 rpm	83 rpm
Weight	72 g	66 g
Resolution	0.088 °	0.088 °
Operating Angle	0-360 °	0-360 °
Max Curren	1.4 A	2.6 A
Price	240 €	105 €

Table 2.1: Comparison board between actuators

2.3 Rear Shaft

If we observe how the original Poppy robot was designed, it needed to follow the screw distribution used by Dynamixel motors (Figure 2.2). This distribution provides more point to fix the motor to the structure compared with a typical servo. Starting from the work completed by Dimitris Zervas, the first step was to make the motor easier to be mounted. With the objective to become an easy platform to be replicated, a case was designed to provide enough fixation points, able to hold the mechanical efforts required and easy to be produced in a standard 3D printer with common hardware.

The case aims to keep the screw pattern of the Dynamixel MX-28AT to become compatible with Poppy parts and reduce the number of modifications needed. The Dynamixel

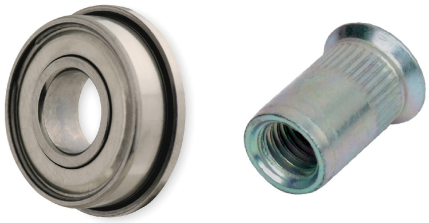

Figure 2.3: Left: Flanged Bearing, Right: M5 Rivet nut

motors also provide a small hole on the opposite face of the gear to attach a rotation-free shaft. This rear shaft offers a second supporting shaft that reduce significantly the moments applied in the main shaft. The first problem that we faced was the lack of a rear shaft in the Hitec servo. The true fact is Hitec has a version of servos with rear shaft and a "Robot Servo" class, but neither of the two provides the required torque. So for all that, the case must include a rear shaft.

There was several solutions for the rear shaft problem but a great number of them demand the use of machined metal, raising the cost and eventually becoming inaccessible for hobby roboticist. This is why the "Bearing-Rivet Nut" alternative has been chosen. This solution consists in a flanged bearing fixed with a rivet nut. Once the rivet has been deformed the pairing is permanent and both elements are concentric.

2.4 Servo case

Once we had determined the requirements, the first version of the case has been designed. The case consists in two pieces, one on the top and the other on the bottom of the servo. The idea is to create a "sandwich" of elements where the servo is in the middle and fix the whole set with four DIN912 M4 screws. The bottom part contains the "Bearing-Rivet Nut" solution.

2.5 Gear Hub

Other problem that we faced was how to fix the body parts of the Poppy platform to the geared axis. The manufacturer provides a part named "arm" where to fix. These arms only have a pair of holes where to insert some bolts, but not for screws. This is why we need to add a hub on the output axis. These lightweight servo hubs are ideal for attaching wheels, gears, or any heavy duty servo horns directly to Hitec servo spline. These aluminium servo hubs have a diameter of 1 inch, and a width of 0.17 inches.

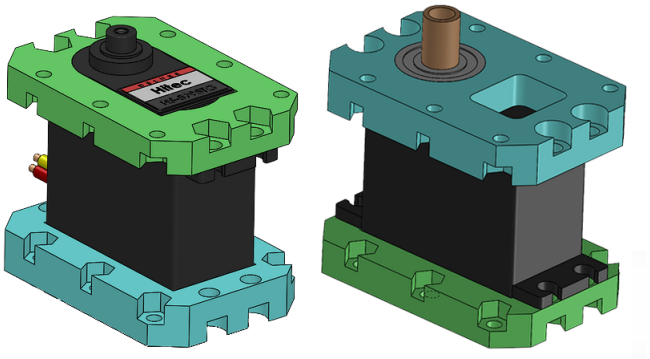

Figure 2.4: View of the top and bottom parts

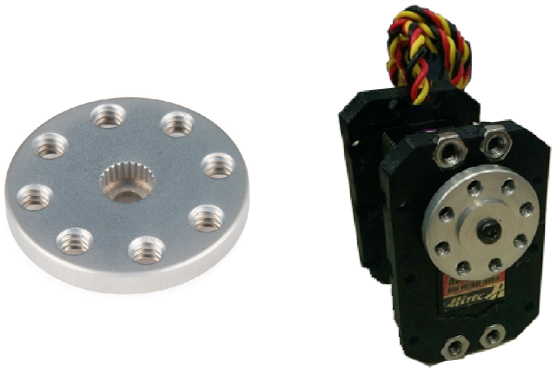

Figure 2.5: Servo HUB and a complete servo with case

2.6 Final design

During the evolution of the master thesis some features of the designed case needed to be changed. The spaces for the hardware that fix both parts have been enlarged. On the top part a small cut has been made to allow a better fit for the servo. A hole for the wires has been made, to avoid in some cases bitting them with the designed parts. The rest of modification tries to fix the differences between CAD model and the real one printed.

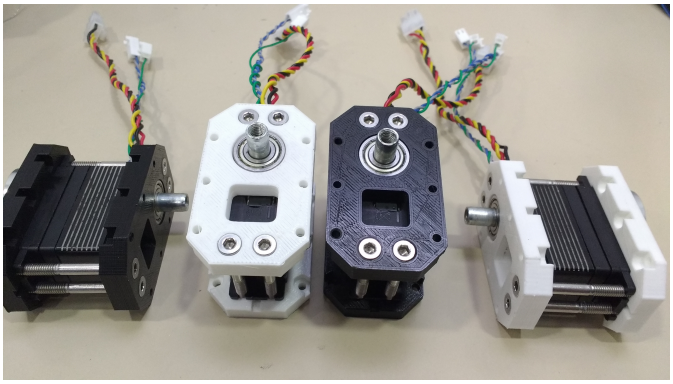

Figure 2.6: Final version of the case

Chapter 3

Design

3.1 Matthieu Lapeyre design

As it has been explained in the previous section, this master thesis focuses on the lower limb of the humanoid robot. which is the interesting part for the bipedal walking. The design that we use as starting point is the one published by Matthieu Lapeyre. The lower limb platform consist in a set of 10 degrees of freedom (5 for each leg) symmetrically placed respect the sagittal plane. One of the main features of the design is the bended thighs.

In his dissertation, Matthieu highlighted the benefits of this option versus a straight thigh. The more separation between the legs, greater the oscillation of the hip walking in a straight line. So this movement is transferred to the upper limb creating additional efforts for the structure. On the other hand the less separation, smaller the stability area, making easier falling down the platform.

3.2 First approach

In the previous section we described the oscillation and stability problems that the structure has. So our new design must provide a better solution for these problems. The alternative was to include an extra degree of freedom on each foot. With this option the distance of the foot can change, providing a more adaptable platform.

Other feature to be worked was how to integrate the 6 actuators placed in the pelvis of the robot. The parts designed for the Dynamixels actuators were too custom and hard to be redesigned for our modified servos. So an alternative was created. In order to connect the actuators in serial we have designed a pair of extra parts. These parts we called as “arms” (see Figure 3.2). With them, the way to fix two actuators become easy and they can be 3D printed without problems.

Figure 3.1: Lower limb of Poppy platform

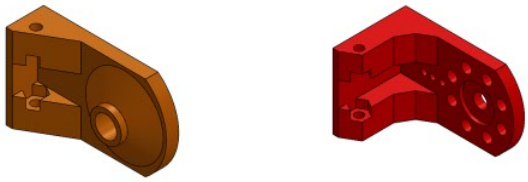

Figure 3.2: Arms set

The next step was to include our modified servos to the existing robot and check what changes were needed. To do that we created the hip with our parts, added the Matthieu's thigh, shin and foot. As a result we got a first idea of our platform.

3.3 Leg modification

As we can see on the first approach, there are some inconsistence with the actuators because they are bigger than the space planned for Dynamixel motors. Therefore, some modifications were needed n the leg parts to fit the new actuators.

The first modification was to made wider the thigh and shin for fitting the case. The location of the screws for the hub changed to adapt to the new one and a new support hole was designed for the rear shaft. The objective in this modification was to keep the silhouette of the original design with the big opening in the front part and the net structure on the rear face.

As it can be seen on Figure 3.4, the thigh is wider and keeps the distance between lateral faces because in lower part must fit our modified servo. The bend distance of both parts is the same, looking for the center of the joint axis. The original thigh has

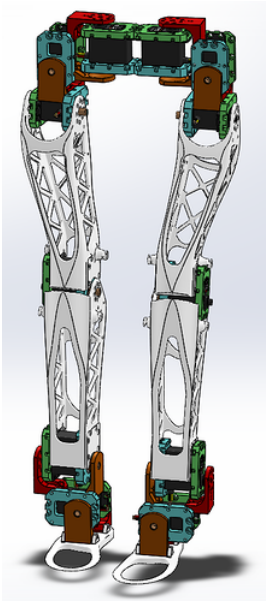

Figure 3.3: First approach of the platform

been made using a SLS printer but for the modified platform a FDM printer has been used.

During the first assemble of the leg, the thigh where the great part of the issues where located. The first one was the blend shape, with few small flat surfaces where start printing, this design required a huge amount of supports. One of the best ways to print it was to place it vertically. Due to the high of the part, not all printers in the market are able to produce it.

Other problems were the tricky way to fix the modified servo on the top joint of the thigh. The original parts include a set of holes for fixing the rear shaft of the Dynamixel. In our case, we offer a shaft wide enough that only needs a hole to fit and the design offers a hole with the just diameter. With some pressure they can be jointed, both elements, and stay fixed.

This fact required to twist the ends of the thigh. Due to the orientation of the filament when the part is being printed, an effort applied perpendicularly could break it more easily. So there is a possibility to damage the structure during the assembly process.

For all this, a second modification was designed. The bend thigh has been changed by a straight thigh to obtain a two flat surfaces. Then it has been split in two parts. The idea is to fix each part of the thigh on the modified motor. The result can be seen in Figure 3.5.

When the benefits of splitting parts were observed, this solution has also been applied

Figure 3.4: Comparison between modified thigh (left) and original (right)

to the shin parts. With that the assemble and subsequent repairs will be easier to be developed. It is true that the platform has lost one of its interesting shapes, but now the part can be printed without complications on any standard 3D printer.

3.4 Feet

During the realization of this master thesis, new versions of the Poppy project where published. One of the features that some users asked for was about the feet. The first version was too simple, so new options appear in the community. All of them were versions that only include one degree of freedom the ankle. So many modifications were required for our project.

In this case we design our version for the feet. This version includes an especial set of parts to adapt with the modified motor shafts. Hence, a total of 3 parts form a single foot. The reason for splitting it was the same as the reasons given in the thigh and shin designs. The foot will be easily put together and to produce in a 3D printer. Other reason is that in a future version of this project, other kind of feet can be attached (i.e. with sensors, different shape or material).

During the iterative process of design and verification, a flat foot becomes easy to twist when the efforts are supported by the edge of the foot. For this reason a set of ribs

Figure 3.5: Comparasion between first modified part (left) and split part (right)

were placed transversely to increase the inertia and bring more rigidity to the foot.

3.5 Upper limb

In the Introduction we explained that this project will focus on the lower limb of the humanoid robot. However, this approach does not mean that we are going to work only on it. In fact the idea is to become as open as possible to future modifications. Hence the upper part must be as modular as possible, this is why we decided to use a standard structural framing system. This system is used widely in the world, so there exist a great

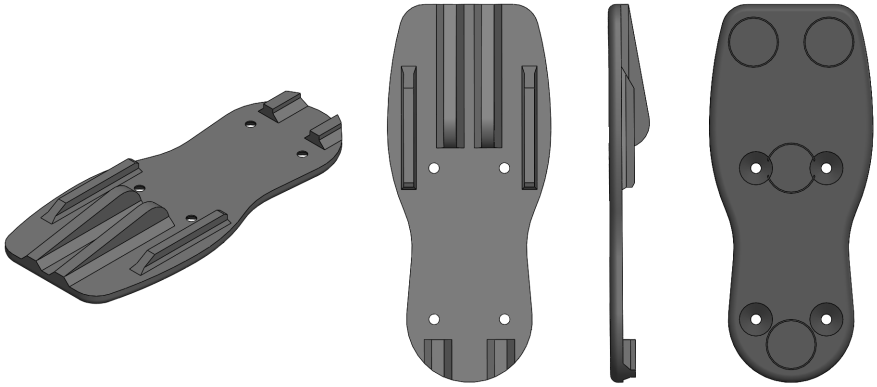

Figure 3.6: Views of the soles of the feet

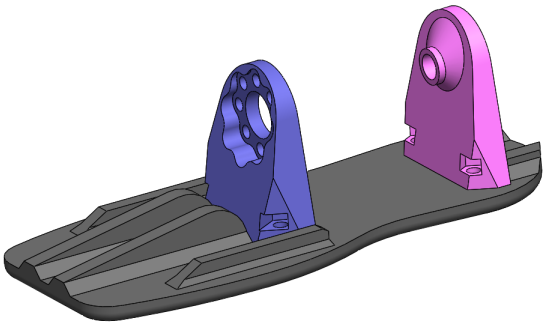

Figure 3.7: Design of the complete foot

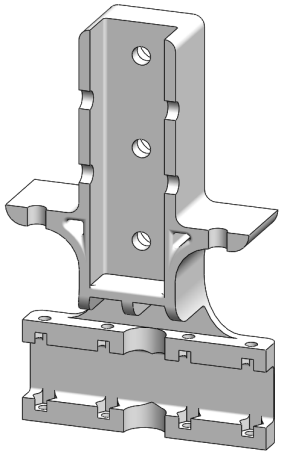

Figure 3.8: One half of the hip design.

number of compatible elements like structures and devices. Moreover, the user can easily fix his/her creations.

The idea is to use a pole of 460 mm of length with a square profile of 20 x 20 mm as the spine of the robot. To fix any kind of element, it is only necessary acquire especial M4 nuts for this frame system called T-nuts. Other option is to find alternatives like print a case for a M4 nut to use it as a t-nut. With all of these elements the users can be able to attach a small computer, power supply or even weight to raise the center of mass of the platform.

The next step is to design a part that fix the top motor of both legs and the spine. What we propose is two identical parts that keep all together using the "sandwich" method. Due to the efforts that this part will suffer, all the possible parts have been reinforced. The result can be seen in Figure 3.8.

3.6 Assembly

Once all the parts has been designed, the last step is to put all them together to create our robotic platform. First on the CAD design, moving all the parts and checking to avoid collisions between parts during the normal operation range of each joint. This procedure helps to avoid situations where the join can be blocked and save time later.

With all the parts verified we start printing the rest of the components. In this case the printers used were the Tumaker Voladora V2 for the initial parts, the Prusa i3 Hephestos for the initial thighs and shin, and finally the BCN3D Sigma for the final version of the parts. All the parts where printed with PLA material. This material is supporting less efforts than ABS (both materials can be found easily in the market) but we keep this in mind during the design, making some parts wider. Other advantage is the fact that PLA offers less resistance at printing, so the number of failures is reduced during the production of the parts. The final result can be observed in Figure 3.9.

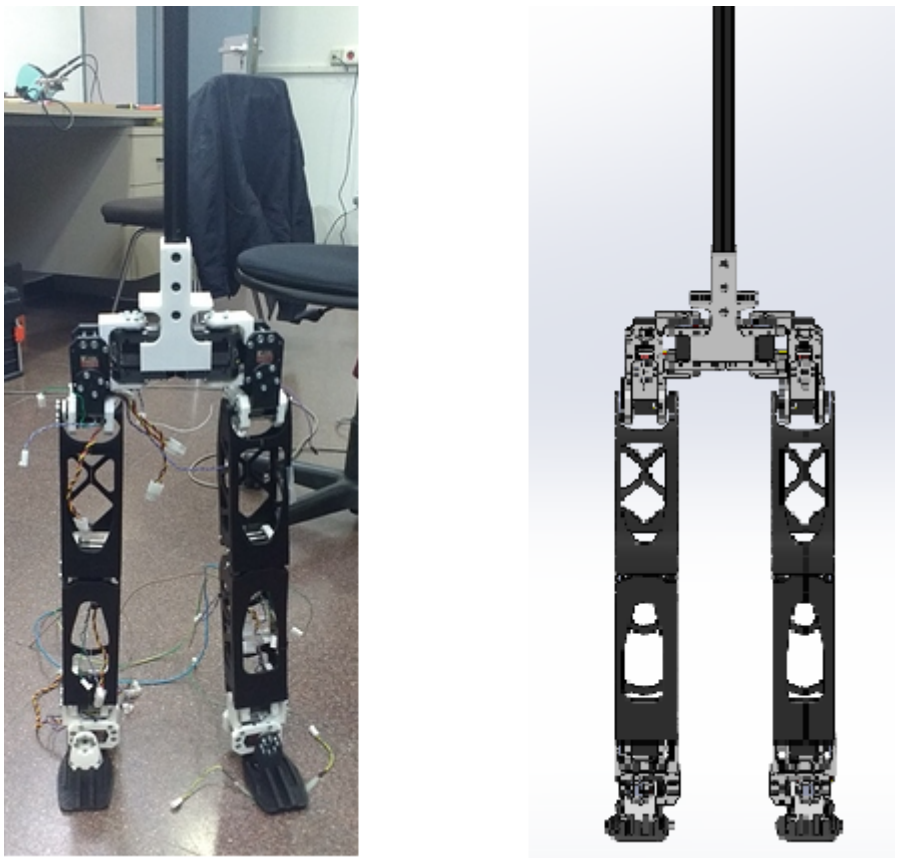

Figure 3.9: Complete assembly (left) and CAD design (right).

Chapter 4

Modelling

4.1 URDF

This Chapter focuses on the physical description of our platform to be used in future simulations. This description can be defined using a wide range of formats. In our case we decided to use the URDF [9] format. The Universal Robotic Description Format (URDF) is an XML file format used to describe all elements of a robot.

4.1.1 Link

In the URDF files, each part of the robot is named "link", these links are composed by a set of elements grouped in 3 tags: `<inertia>`, where are described the physical properties; `<visual>`, where the look is defined (just for the comprehension of the user); and `<collision>`, where the shape of the robot and surface properties are explained for the computation of the interaction with other links or environment. As example, below we have a link description in URDF.

```
<link name="my_link">
  <inertial>
    <origin xyz="0 0 0.5" rpy="0 0 0"/>
    <mass value="1"/>
    <inertia ixx="100"  ixy="0"  ixz="0" iyy="100" iyz="0" izz="100" />
  </inertial>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <box size="1 1 1" />
    </geometry>
```

```
    <material name="Cyan">
      <color rgba="0 1.0 1.0 1.0"/>
    </material>
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <cylinder radius="1" length="0.5"/>
    </geometry>
  </collision>
</link>
```

There are more tags to be used to describe a link, but the ones shown in the example are the main ones. In fact, depending on the software which we are going to use our URDF file, we can include more tags. Each software parses the file and keep the information that is relevant for it. In this way, an URDF file can be compatible with all the software and also include unique information for each lonk. In Figure 4.1 we can observe the idea of each one of the tag groups.

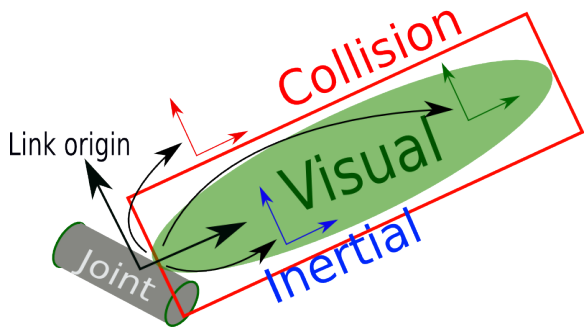

Figure 4.1: Link description visualization.

4.1.2 Joint

Another important component in the URDF of our robot is the "joint" tag. With this tag we can define the relationship between two links. This component not only includes the rotation joints, it also includes other features like fix, revolution, linear or planar. Other information that can be included is the physical limits of the joint and other dynamical properties as can be seen in the example below.

```
<joint name="my_joint" type="floating">
   <origin xyz="0 0 1" rpy="0 0 3.1416"/>
   <parent link="link1"/>
```

```
    <child link="link2"/>

    <calibration rising="0.0"/>
    <dynamics damping="0.0" friction="0.0"/>
    <limit effort="30" velocity="1.0" lower="-2.2" upper="0.7" />
    <safety_controller k_velocity="10" k_position="15" soft_lower_limit="-2.0
    soft_upper_limit="0.5" />
</joint>
```

Some considerations should be taken when joining joint and link. As it can be observed in Figure 4.2, there is displacement of the coordinate axis of each joint and link. As agreement, the first element always is the "base_link" and the origin frame is the coordinate frame. When we attach a joint to this link, the link is defined as the "parent". The coordinate frame is the same as the origin frame of the parent link. So the joint origin should be located as well as the orientation of the axis.

The same situation happens when we attach the next link to the joint. In this case the link is defined as "child" link for the joint. As usual, the coordinate frame of the child link is the same as the origin frame of the joint.

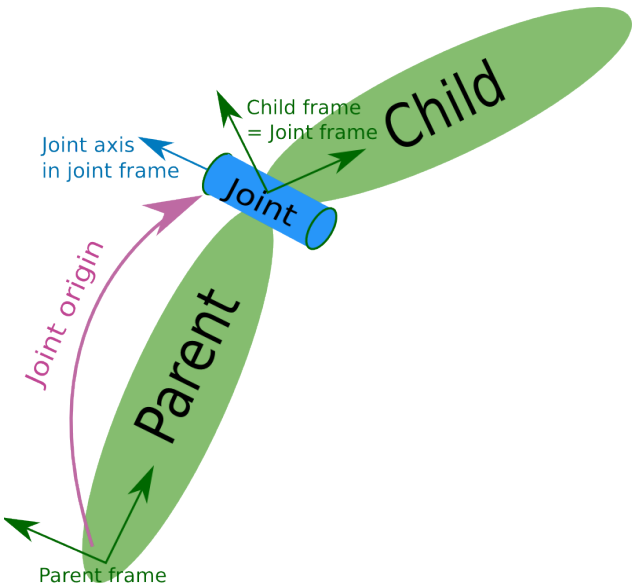

Figure 4.2: Joint description visualization.

4.2 Box model

Once we got the necessary elements, we started to create our first approach. The goal was to create a model of our robot made by boxes. Doing that we were able to check easily all the distances used. These distances can be obtained using the CAD design, so

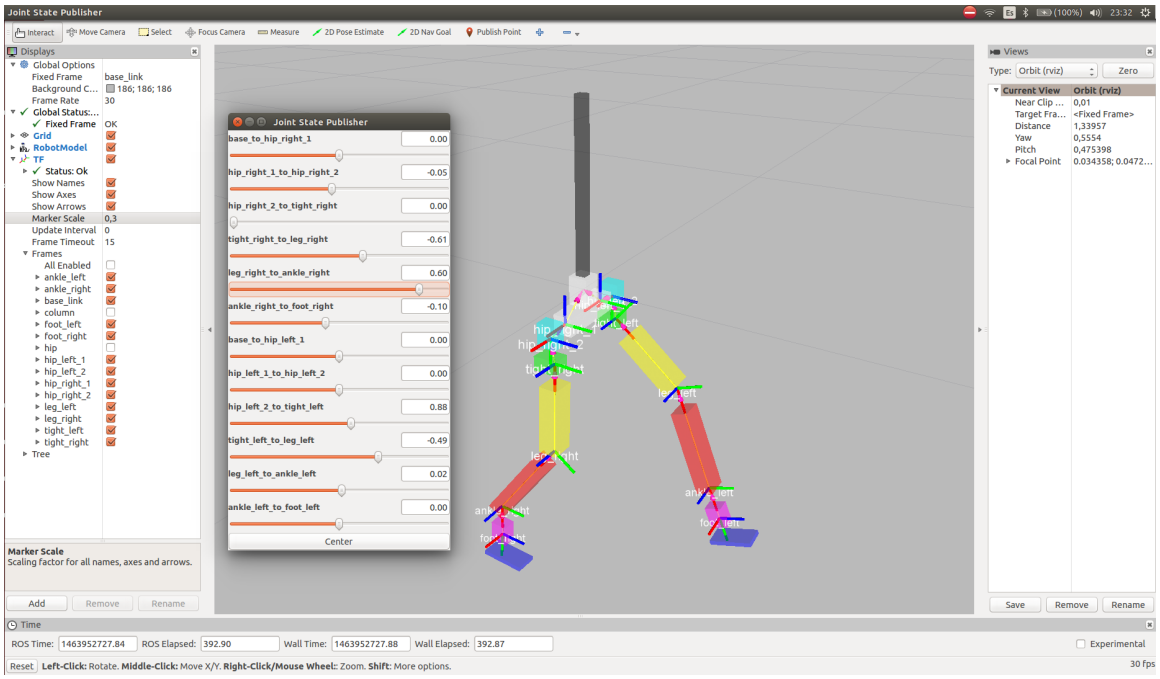

Figure 4.3: Visualization of the box model.

the first step was to define the origin frame of the robot. This origin is located in the hip of the robot, that named “base_link”, included in the lateral and sagittal plane. Fixed to the “base_link” we attached the spine with the physical properties of the real bar.

The next step was to include the top joints of each leg. Due to the existence of 6 actuators in the pelvic zone (3 per leg) we named them using the following rule: “human part” + “side” + “numeration” (if needed). So we created, for instance, the link “hip_left_1” that is related to the hip using the joint “base_to_hip_left_1”. With this nomenclature anyone can identify which are the links that each joint is referring.

With the possibility to define by us the origin of each joint, we decided to follow a simple rule: “The origin of each joint is located in the outer side of the rear shaft and is centred in the axis of the motor”. With this rule, we were able to obtain a first descriptor of our robot using the URDF structure. Once we got the robot defined, we created the visual part of the robot with boxes. This model made with boxes only serve as a validation method as can been seen in Figure 4.3.

To be able to move the robot as a puppet an see the visualization we need to adapt our work to the desired tools. These tools are defined in the following.

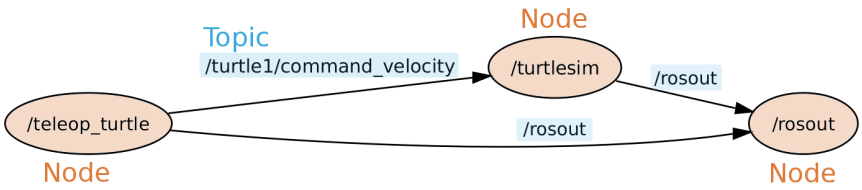

Figure 4.4: Example of node communication.

4.3 ROS

What we needed to be able to interact with the model of our platform were a set of tools. From all the existing options we decided to use ROS, that comes from "Robot Operative System". ROS can be defined as a framework for robotic development that provides a huge set of tools, programming libraries and useful console commands. Despite of the name, ROS is not a operating system, in fact it runs over Unix.

To be used as a universal framework, ROS requires a hardware abstraction in the way that each platform can understand the same commands. As example, if ROS orders move ahead, the robot must follow without matter if the platform is a differential, biped or flying robot. Thanks to that, we can use the more complicated algorithms provided by ROS for high level purposes as navigation or exploration.

What makes ROS an extended tool in the automation community is the distributed communication between nodes. A node is any script of code, the more simple, the better. Using nodes forces to do a abstraction exercise and use some rules of communication with other nodes. This way helps when the groups of people needs to work together and be able to reuse them.

The nodes communicate between them using the elements provided by ROS. The main elements are "Topics", "Publishers" and "Subscribers". A "Topic" is a channel where the messages flow. These messages contain data in a structured way. Each node can access to any "Topic" in two ways. The node can publish new data to the "Topic", the node use the "Publisher" elements of ROS. Either, the node wait for new data in the "Topic" to act, in this case the node becomes a "Subscribe"" of a "Topic".

In Figure 4.4 can be identified 3 nodes and 1 topic. The direction of the arrow defines which node is the publisher of the topic, and who has subscribed to it. In general, a simple node is not enough to perform a task, so a set of them are created to achieve a common goal. What ROS offers is to group these nodes and files in folders with a defined structure. These structures have been named as "Package". This method helps to keep a clean solution and make them compatible with the "Packages" of other authors. All of

them have following basic form.

```
my_package/
  CMakeLists.txt
  package.xml
```

On one hand, the `CMakeList.txt` is defined with all the information for the compiling process of the nodes, the definition of the data inside the messages and the publishers and subscriber nodes. On the other hand, the `package.xml` includes the information as the authors, the kind of license or the dependency with other packages. Depending on each package, it will include more folders and files.

4.4 Poppy_upc_description package

To be our URDF model the more compatible with future ROS users we decided to design a set of packages following the recommendations of the community. As a non-spoken rule, the URDF files must be located in the "description" package of our robot and must include the following structure:

```
poppy_upc_description/
  launch/
  meshes/
  msg/
  src/
  urdf/
  worlds/
  CMakeLists.txt
  contributors.txt
  package.xml
```

The `launch/` folder includes the XML files with the `.launch` extension. These files include a set of parameters and nodes to be called. Hence, when running a file, we can have different nodes working at the same time. As we can observe in Figure 4.3, there is a window with some slide controls. This node is provided by other package called "urdf_tutorial". To use it, we include this node in the launch file with other nodes.

```
<launch>
      <arg name="model" />
      <arg name="gui" default="True" />
      <param name="robot_description"
      textfile="$(find poppy_upc_description)/urdf/poppy_upc.urdf" />
      <param name="use_gui" value="$(arg gui)"/>
      <node name="joint_state_publisher" pkg="joint_state_publisher"
```

```
      type="joint_state_publisher" />
      <node name="robot_state_publisher" pkg="robot_state_publisher"
      type="state_publisher" />
      <node name="rviz" pkg="rviz" type="rviz" args="-d
      $(find urdf_tutorial)/urdf.rviz" />
</launch>
```

The `meshes/` folder includes the shapes of the links and textures from the CAD files. This is useful when the link has some shape that is not possible to define with a set of simple geometries (like boxes and spheres). In our case, we exported each one of the parts of the platform in `.stl` format. The STL (STereoLithography) is a file format native to the stereolithography CAD software created by 3D Systems. STL has several after-the-fact backronyms such as "Standard Triangle Language". This file format contains a mesh defined only by triangles and is widely used for rapid prototyping, 3D printing and computer-aided manufacturing. The last step is to change the geometry tag...

```
<geometry>
  <box size="0.15 0.04 0.04"/>
</geometry>
```

...by the mesh import.

```
<geometry>
  <mesh filename="package://poppy_upc_description/meshes/hip_right_1.stl"/>
</geometry>
```

The result can be seen in Figure 4.5.

The `msg/` folder includes the text files with the `.msg` extension. This file includes the type of data and the correspondent name. In our case, we created our message type file. Our files include the position, velocity and offset of the 12 actuators.

```
float32[12] position
int16[12] velocity
int16[12] offset
```

This `.msg` file must be in all the packages that we want to use (and modify the respective `CMakeList.txt` and `package.xml` files.

The `src/` folder includes all the nodes of the package. These nodes can be written in C++ or Python language. Even nodes with different languages can communicate between them.

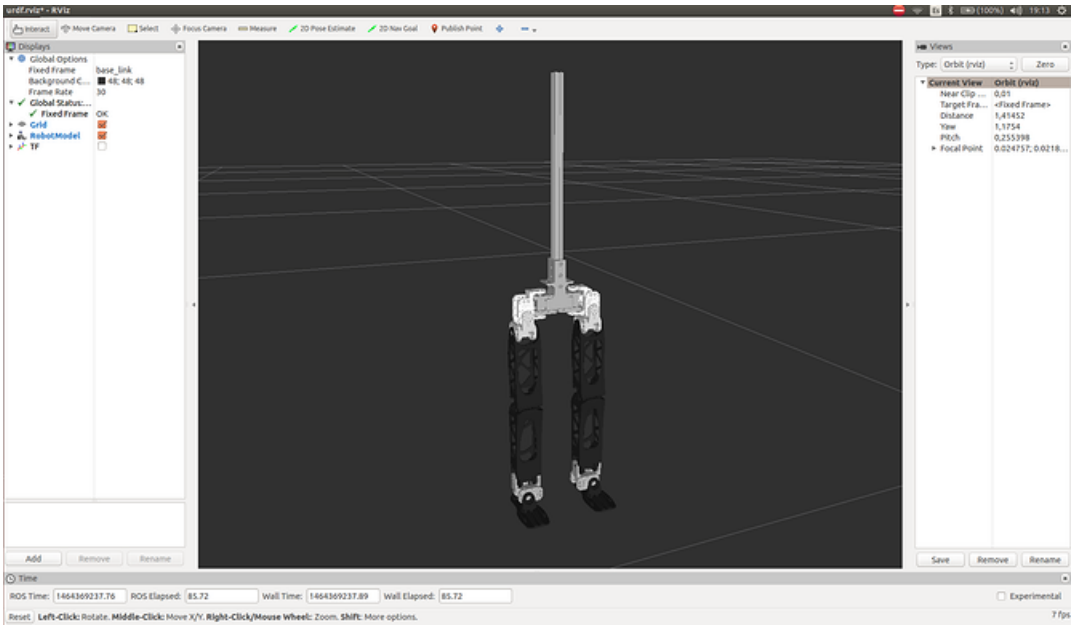

Figure 4.5: Visualization with the real meshes.

The `urdf/` folder includes the XML files with the `.urdf` extension used to define a robot.

The `world/` folder includes the XML files with the `.world` extension. These files define the properties of a world in the case we want to simulate the physics of our robot in a virtual environment. Several simulators can be used, but the most used with ROS is the Gazebo simulator.

4.5 Poppy_upc_gazebo package

ROS is compatible with the most used simulators. In this master thesis we focused on the use of Gazebo. A well-designed simulator makes it possible to rapidly test algorithms, design robots, and perform regression testing using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments.

In the previous section we explained the files with the `.world` extension. For simple simulations this file is enough, but for our case is not. Due to the several parameters needed to define the simulation as real as possible, an extra package has been created. This package only include two folders, `launch` and `worlds`.

In the `launch` folder we placed the `poppy_upc_world.launch` file. This file includes another `.launch` file from the "gazebo_ros" package with the basic structure of the a

environment, a `.world` file from our package and the description of our platform.

```
<?xml version="1.0"?>
<launch>
<include file="$(find gazebo_ros)/launch/empty_world.launch">
<arg name="world_name"
    value="$(find poppy_upc_gazebo)/worlds/poppy_upc.world"/>
<arg name="paused" value="true"/>
<arg name="gui" value="true"/>
</include>

<param name="robot_description" command="$(find xacro)/xacro.py
    '$(find poppy_upc_description)/urdf/poppy_upc.xacro'" />

<node name="mybot_spawn" pkg="gazebo_ros" type="spawn_model" output="screen"
      args="-urdf -z 0.608 -param robot_description -model poppy_upc" />
</launch>
```

In the `world` folder we have the `poppy_upc.world` file. This file includes the same basic parameters as other `.world` files, but also includes the parameters for the solver of the physics. Each solver provides better results depending on the application. In our case we use the same solver used by the company PAL Robotics in the files published for the robot REEM C. Then, we adjust some parameters to obtain the desired behaviour. Depending on the number of iterations, the step time, and other values, the result can become unstable because the solution for each step has not been reached. Below we show the `physics` tag with all the parameter.

```
<physics type="ode">
  <gravity>0 0 -9.81</gravity>
  <ode>
    <solver>
      <type>quick</type>
      <iters>50</iters>
      <sor>1.0</sor>
    </solver>
    <constraints>
      <cfm>0.0</cfm>
      <erp>0.2</erp>
      <contact_max_correcting_vel>100.0</contact_max_correcting_vel>
      <contact_surface_layer>0.0</contact_surface_layer>
    </constraints>
  </ode>
```

```
  <real_time_update_rate>1000</real_time_update_rate>
  <max_step_size>0.001</max_step_size>
  <max_contacts>20</max_contacts>
</physics>
```

Chapter 5

Control

In this project we defined the *Control* part as all the elements created to be able to manipulate our platform. The control algorithm must be developed by the future users, but some examples are provided to facilitate the work. The control part has been divided in three sections. The first one includes all the necessary to work with the simulator, the definition of the joints that can be controlled (in our case, all of them) and the controller that is being used.

The other two sections include the files to work with the real platform. This task requires the definition of two types of controllers. The global one, located in the CPU unit that communicate with all the actuators and run at 50 Hz, programmed as a ROS package. The second controller is the firmware of the motors. Each motor runs a control loop at 500 Hz. The control parameter of these two parts can be changed at any time.

5.1 Poppy_upc_control package

Following the recommended structure by the ROS community, the control files must be placed in a different package. With this package we can define a different control for each joint. The typical use is the definition of the PID values. The structure of this package is:

```
poppy_upc_control/
  config/
  launch/
  CMakeLists.txt
  contributors.txt
  package.xml
```

The `config/` folder includes the files with the `.yaml` extension. In our case, the file is divided in two parts. The first one defines which global controller will be used and which

refresh rate will work. The idea here is to define the frequency of the closed loop for each joint. The second part defines which type of control we are using (position or velocity in general) and the PID values.

```
poppy_upc:
  # Publish all joint states -----------------------------------
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50

  # Position Controllers ---------------------------------------
  joint1_position_controller:
    type: effort_controllers/JointPositionController
    joint: base_to_hip_right_1
    pid: {p: 100.0, i: 0.01, d: 10.0}
```

In this case the `launch/` folder includes the `.launch` files to define the sequence of nodes and parameters needed to simulate our robot with the available controls. As it is shown below, the first part of the `.launch` files defines the control parameters of the `.yaml` file. Then, the controller is loaded of each one of the 12 joints. Next, it uses the publisher node that ROS provides. Finally, points to the robot description, the `.xacro` file.

```
<?xml version="1.0"?>
<launch>

  <!-- Load joint controller config. from YAML file to parameter server -->
  <rosparam file="$(find poppy_upc_control)/config/poppy_upc_control.yaml"
  command="load"/>

  <!-- load the controllers -->
  <node name="controller_spawner" pkg="controller_manager" type="spawner"
    respawn="false"
    output="screen" ns="/poppy_upc"
    args="joint_state_controller
joint1_position_controller
joint2_position_controller
..."/>

  <!-- convert joint states to TF transforms for rviz, etc -->
  <node name="robot_state_publisher" pkg="robot_state_publisher"
    type="robot_state_publisher"
    respawn="false" output="screen">
```

```
    <param name="robot_description" command="$(find xacro)/xacro.py
      '$(find poppy_upc_description)/urdf/poppy_upc.xacro'" />
    <remap from="/joint_states" to="/poppy_upc/joint_states" />
  </node>

</launch>
```

Once we have defined the controlled joints, we can publish the desired goal of each joint using a "Topic". The user can create his/her own package or modify ours, then just has to program a node as a "Publisher" to push the data to the controllers. The parameters provided are not tuned, so if the launch file of this package is run, the result of the simulator will start oscillating and finally let the body hit the floor. As explained before, in case to want control the real platform, the next packages must be used.

5.2 Motor_driver package

This package was created by Dimitris Zervas during his master thesis. In this form, any user can easily communicate with a set of connected motors. The examples provided allow to the user change the desired velocity of each motor using a very modular protocol. Thanks to this protocol, we were able to add to the "motor_driver" package the possibility of sending position goals and still being able to send velocity goals. So no features were lost. The goal that the motors will follow depends on the firmware of the motors, the protocol also offers the possibility to change online the controller (from velocity control to position control). Some modifications have been made from the original package, the new structure is:

```
motor_driver/
  config/
  include/
  launch/
  msg/
  src/
  CMakeLists.txt
  contributors.txt
  package.xml
```

As in the previous package, the `config/` folder includes the configuration files in `.yaml` extension. In our case due to the fact that the reference of the motor depends on the manufacturing process, we need to define a offset. Another important configuration is the direction of rotation of each motors. Depending on the position in the platform, the reference axis of rotation can be inverted respect the URDF description file. As an example, the offset file has this form:

```
motors_offsets:
# The offset values of each motor.
# Offset values are in ticks

# RIGHT LEG
    joint1:
        name: base_to_hip_right_1
        offset: 3264
        direction: 1
```

The `include/` folder is from the original package and has all the external data and files like documentation.

The `launch/` folder contains some files related with the configuration `.yaml` files, in order to include them in especial cases. Nowadays each modified servo includes the set of offsets, so this feature is not used, however it can be useful for future users.

The `msg/` folder contains a set of messages definitions. In our case we created the `Full_feedback.msg` file. This file must be exactly the same as defined in the other packages.

5.2.1 Topics setup

Finally, the `src/` folder contains the scripts where all the hard work is done. These scripts are between the ROS infrastructure (software) and the serial communication with the motors (hardware). Aside from all the included libraries, variable definition, and function declaration. The definition of the topics used are crucial. In the code below we show the first part of the main loop:

```
int main(int argc, char **argv) {

    ros::init (argc, argv, "serial_test");
    ros::NodeHandle nh;
    signal(SIGINT, mySigintHandler);  // to handle ctrl-c

std::ofstream myFile;

com.start();
usleep(1000000);
// ROS_INFO("Wait!");
// ------------------------ Publishers -----------------------------//
ros::Publisher motor1_pub = nh.advertise<motor_driver::Feedback>(
```

```
    "motor_driver/motor1_feedback", 1000);
ros::Publisher motor1_vel = nh.advertise<motor_driver::Velocity>(
    "motor_driver/motor1_velocity", 1000);
ros::Publisher motor_feedback = nh.advertise<motor_driver::Full_feedback>(
    "motor_driver/motor_feedback",100);
motor_driver::Feedback motor1_feed;
motor_driver::Velocity motor1_v;
motor_driver::Full_feedback ffb; //FullFeedBack (ffb)

// ------------------------- Subsricers ----------------------------//
ros::Subscriber motor_vref = nh.subscribe<motor_driver::Goals>(
    "motor_driver/motor_setGoal", 1000, &callback_setGoal);
ros::Subscriber motor_goalPos = nh.subscribe<motor_driver::Goal_Pos>(
    "motor_driver/motor_setPos", 1000, &callback_setPos);
ros::Subscriber motor_goalVel = nh.subscribe<motor_driver::Goal_Vel>(
    "motor_driver/motor_setVel", 1000, &callback_setVel);
ros::Subscriber motor_Gains = nh.subscribe<motor_driver::Gains>(
    "motor_driver/motor_Gains",1000, &callback_setGain);

nMotors = com.update_motors_list(motors_list);
ROS_INFO("Found '%x' motors", nMotors);

ros::Rate loop_rate(70);
```

Besides the ROS initialization, the instruction `com.start();` enables the serial communication with the protocol made by Dimitris Zervas. Next, the publishers are defined. In our case we included the "Full_feedback" topic as a publisher. On the other hand, based with the existing topics, we created the "Goal_Pos" topic as a Subscriber. Then, using the instruction `nMotors = com.update_motors_list(motors_list);` we check the number of detected motors and show the info to the user. Finally we define the frequency of the execution of the ROS loop. In the folowing subsections we show examples of the second part of the node, depending which use we want to do. Also both uses can be done in the same loop.

5.2.2 Read data

In this second part we expose how to read data from the modified servos. In our case we want to read the position and velocity of each one of the motors detected during the initialization, then we save these values in a the message topic. One we have all the data, we publish it using the correspondent topic. Finally we leave the execution idle, waiting to close the loop in the required time. The associated code is:

```
    while(ros::ok())  {
```

```
        for (uint8_t i=0; i<nMotors; i++) {
            motor1 = com.get_position(motors_list[i],position);
            pRad = position*6.2932 / 4096;
            motor1 = com.get_velocity(motors_list[i],velocity);
            ffb.position[i] = pRad;
            ffb.velocity[i] = velocity;
        }
        motor_feedback.publish(ffb);

        ros::spinOnce();
        loop_rate.sleep();
    }
}
```

5.2.3 Send goals

Another option is to change the desired goal in each one of the motors. Each motor will receive the updated goal and use it during the control loop of the firmware. It can be observed in the code below:

```
    while(ros::ok())  {

        pos_reference = 0.0; //comment for subscriber use
        goal_pos = (int16_t)(pos_reference*4096.0/6.283185+2048.0);
        if (goal_pos>4095){
            goal_pos = 4095;
        } else if (goal_pos<0){
            goal_pos = 0;
        }
        for (uint8_t i=0; i<nMotors; i++) {
            if (motors_list[i]!= 0) {
                com.goal_position(motors_list[i],pos_vect[i_vect]);
                ROS_INFO("Goal position send to %x:", motors_list[i]);
            }
        }

        ros::spinOnce();
        loop_rate.sleep();
    }
}
```

In this example, the variable `pos_reference` is always `0.0`, just to fix the robot in the

stand position. Then, we compute the transformation from radians (that is the universal unit used by ROS to define the rotation of an axis) to `ticks`, because the position encoder divides the circumference in 4096 positions or `ticks`. Once we have computed the new goal in `ticks` for each motor, we transfer them using the `com.goal_position` node.

In the case that we want to use the subscribed topic of the goal position, we just need to comment the first line inside the ROS loop and treat it as a vector. To be able to update this variable each time someone is publishing in the "Goal_Pos" topic, besides from initialize the subscriber, we define this function to store the data in the `pos_reference` variable.

```
void callback_setPos(const motor_driver::Goal_Pos::ConstPtr& msg)
{
    pos_reference = msg->goal_position;
}
```

5.3 Servo firmware

Since this platform is programmable using the Arduino IDE, it allows to easily develop any kind of control algorithm. Moreover, when in a worry, a huge community is behind this tool, so new users have a huge amount of information outside.

This Section is divided in two parts: configuring the firmware, to be performed just once, and programming, where we show how to develop the controller.

5.3.1 Initial configuration

To be able to program the control loop inside each one the motors, an initial setup is required. So some modifications must be completed:

- Include the file `ATmegaBOOT_168_uno20.hex` inside the `bootloaders/atmega` folder. Hence, we will be able to load the boot firmware in case we used a fresh-new chip.

- Modify the file `boards.txt` inside the `hardware/arduino/avr` folder. In that file, the last lines must include the parameters of the board at 20Mhz. So the last lines provided in our files must be included.

Once we completed this, we open the Arduino IDE, go to the label `Tools`, click on `Boards` and select `Arduino @ 20Mhz`. From this point we will able to develop future control algorithms.

5.3.2 Motor_poppy sketchbook

The code inside the modified motors is composed by 3 files:

```
motor_poppy/
  CRC.ino
  fill_buffer_tx.ino
  motor_poppy.ino
```

The `CRC.ino` file includes the error correction parameters for communication. This file must be not modified, but it is necessary for the compilation of the firmware.

The `fill_buffer_tx.ino` file contains the definition of the messages for the communication. In case that a new one kind of data message is needed, then a new type can created. For our project we developed the `GOAL_POSITION` message as case `0x02`.

```
    case 0x02:                                  // GOAL_POSITION
    {
      int *p = (int *)&bufferRx[5];
      goal_position = *p;
      bufferTx[3] = 0x02;
      bufferTx[4] = error;
      num = _crcTx(bufferTx, 0x02);
      write_flag = true;
      break;
    }
```

The code reads the starting 4th byte of the sequence ans stores in the `goal_position` variable, previously defined in `motor_poppy.ino`. The next line check the type of message as well as some error is present.

The last file, `motor_poppy.ino`, is the heart of the firmware. This file includes all the functions to communicate with the computer, sensors and motors. There exist two parts where the user must work, the rest cannot be modified.

At the beginning of the file, we can find these lines

```
#define ADDR    0x01
...
int16_t offvect[12] = {3264, 3403, 3971, 1837, 1062, 760, 26, 1513, 3602,
                       145, 1311, 2664};
```

`ADDR` defines the identification number and must be unique. This value is used to identify the motor offset. All the offset is saved in `offvect[]`. The value inside this vector can be modified using the communication tools by computer, but this way allows to set the position at the power up, before the communications starts.

```
void loop() {
  if (control_loop == true) {
    /* ----------- CONTROLER CODE HERE ------------ */
    e_pos = goal_position - pos_axis;
    if (abs(e_pos) < goal_tolerance) {
      e_pos = 0;
    }
        // P-term
    uP = Kp * (float)e_pos;

    // I-term
    uI = uI + Ki * (float)e_pos;
    if(abs(uI)>255.0){
      if(uI>0.0){
        uI = 255.0;
      }else{
        uI = -255.0;
      }
    }

    // D-term
    uD = Kd * (float)e_pos - Kd * (float)prev_e_pos;

    // PID output
    u = uP + uI + uD;

    ...
    }
  ...
}
```

The control loop can be found inside the `void loop()` function. In this case we show a PID example of code and requires some considerations:

- The control parameters are defined outside this function.

- The position reference includes some tolerance, just to avoid oscillation in the goal.
- The I-term output of the controller is saturated at 255.
- The control output is saturated when it becomes larger than 255 since value is translated to 8 bits PWM resolution of the control signal.

Chapter 6

Costs

6.1 Time

This project has been completed in a period of 24 weeks. During this time the design, modelling and control phases has been developed. Since the phases are not independently each from other, there exists some overlap in time. This fact was caused because the interrelation between them. As example, the meshes of the CAD parts have been reduced to minimize the computational cost. In other situations, the shipping time of some elements made necessary to work in other parts to use the time.

About the design part, this was the longest phase because we included the manufacturing process. Although the design and fabrication process can be considered separately phases, the 3D printing technologies allow to become them part of the iteration process during the design phase. This is why this phase cover from the beginning of the project until the complete production of the platform.

The Modelling phase could be done in less time that showed in the Gantt diagram. However, some of the necessary information needed to continue the project was extracted from the community forums. This fact made to redo the previous work done to follow the best structures. Another of the main problems was the gazebo parameters of the simulator. The heuristics process was not really effective due to the quantity of parameters. Thanks to the public packages of another platform, this work was finished.

The last phase was the Control phase and almost requires the halt of the master thesis duration. One of the main task was to adapt the “motor_driver” package to the new platform. During the assembly some control parts were tested, hence helping to find some errors in the initial configurations and in the wiring setup. Once the ending steps were made, a final check of the whole platform was made to repair the last issues.

In the Gantt diagram of Figure 6.1 we can observe how the time was spent in the different sub-task and the relationship between them.

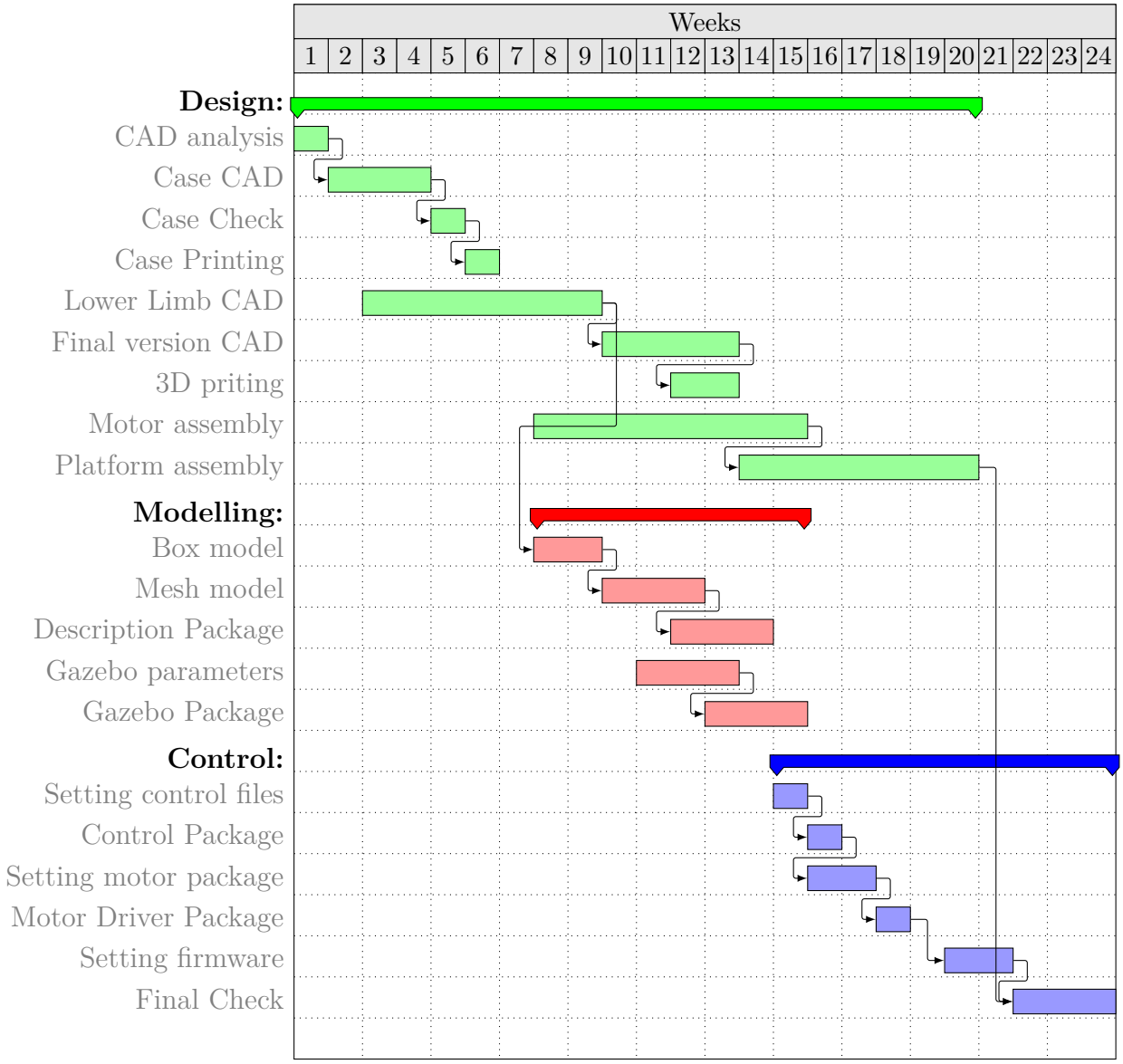

Figure 6.1: Gantt diagram

6.2 Budget

In this master thesis the budgetary cost is divided in two: the material and the personal cost. We made this differentiation just to be able to compare with the original Poppy robot project. Since Poppy must be assembled, this task can be added in the cost of the project. This will be useful to compare later the original project and this master thesis. The total cost of the project grows to 11,390€. Now we are going to break it down.

Concept	Units	Unit price	Cost
MODIFIED SERVOS	**12**	**113.48**	**1,361.76**
Servo HITEC HS-7954SH	12	95	1,140
Modification components	12	10	120
Hub	12	3.3	39.6
Bearing F697ZZ	12	2.7	32.4
M5 Rivet Nut	12	0.15	1.8
Case	12	1.5	18
DIN912 M4x40	48	0.1	4.8
NUT M4	48	0.03	1.44
NUT M3	144	0.01	1.44
Power Connectors and crimps	12	0.12	1.44
UART Connectors and crimps	12	0.07	0.84
POPPY UPC STRUCTURE	**1**	**349.2**	**349.2**
Printing "ARMS"	6	1.8	10.8
Printing "Thigh"	2	48.5	97
Printing "Shin"	2	40.6	81.2
Printing "Foot"	1	15.4	15.4
Printing "Hip"	2	18.5	37
Aluminum Bar 20x20	1	12.3	12.3
DIN7985 M3x20	56	0.08	4.48
Wire	1	7.9	7.9
Power Connectors and crimps	14	0.13	1.82
UART Connectors and crimps	14	0.05	0.7
PCB	1	15.6	15.6
Power Supply 7.4V 30A	1	65	65
TOTAL			**1,710**

Table 6.1: Detailed material cost

6.2.1 Material Cost

The material cost invested in this project has been 1,710€. In this subsection, we want to divide the cost of replacing the actuators and manufacturing the new platform. One important fact is that actuators represent 80% of the cost. In the future, actuators will become more affordable, making this project even more viable.

6.2.2 Personal Cost

Personal cost represents 85% of the total cost for the master thesis. But once we produce a second platform, this cost for the second unit will be only the assembly cost, that is less than 500€. The same structure that we used along the master thesis is used in this break down. One of the interesting issues is that more that the half time spent was in the design phase. Talking about the hour cost, we have divided in 3 ranges, according to

Concept	Hours	Hour price	Cost
DESIGN	**238**		**5,510**
CAD analysis	16	25	400
Case CAD	52	20	1040
Case Check	14	15	210
Case Printing	4	20	80
Lower Limb CAD	88	25	2200
Final version CAD	34	25	850
3D priting	8	20	160
Motor assembly	14	15	210
Platform assembly	24	15	360
MODELLING	**62**		**1,360**
Box model	8	25	200
Mesh model	16	25	400
Description Package	12	20	240
Gazebo parameters	16	20	320
Gazebo Package	10	20	200
CONTROL	**120**		**2,810**
Setting control files	10	20	200
Control Package	14	20	280
Setting motor package	28	25	700
Motor Driver Package	14	20	280
Setting firmware	22	25	550
Final Check	32	25	800
TOTAL	**420**		**9,680**

Table 6.2: Detailed personal cost

the responsibilities and difficulty.

Environmental impact

This part of the master thesis focuses on all the environmental impacts associated to the platform itself. The impact of developing design, modelling, control or even documenting time is not considered.

The first impact to consider is that caused by the transportation of the materials to our lab. As we defined before, one of the objective is to provide a cheaper platform compared with the Poppy Project. About that, we assumed that the impact of transportation is the same for the original actuators or the modified servos. The same can we applied to the hardware. Most of these elements are deadless metals to the environment once have been used, and they can be recycled.

The rest of the impact can be found on the production method, to be specific, on plastics components. Due to some plastics can be toxic to the environment, it is something to analyse. The plastic used in this project is polylactic acid or polylactide (PLA), a biodegradable thermoplastic aliphatic polyester derived from renewable resources, such as corn starch (in the United States and Canada), tapioca roots, chips or starch (mostly in Asia), or sugarcane (in the rest of the world). The PLA decomposes in a short term of 2 years. In 2010, PLA had the second highest consumption volume of any bioplastic of the world.

The original Poppy is printed using the SLS technology, this uses powder Nylon. Nylon descomposes in 30-40years. With the FDM technology, we can print with ABS, a more resistent material than PLA but non biodegradable plastic that needs more than 100 years to degrade. All the solutions [11] (PLA, Nylon, or ABS) can be recicled, but the PLA is the one that requires less energy to produce it.

Conclusions

This thesis has proven the viability of the Poppy UPC platform as suitable for development of bipedal walking algorithms. To being able to confirm that, we review each ones of the proposed objectives and the obtained result.

The greatest challenge was to reduce the cost of the robot. The lower limb of the Poppy project has a cost around 4,200€, 2,600€for the actuators, and 1,600âĆň for manufacturing costs using SLS technology. By contrast, the Poppy UPC platform costs around 1,700€, a reduction of the 60% of the price. In the future this platform can become even more affordable if the servo modified is changed by a cheaper one. Hence, it is possible to become an under-thousand platform.

The design of a case for the modified servos has made them to become more modular and easy to use in other platforms (not only for bipedal robots). Moreover, thanks to the work by Dimitris Zervas, they can be used from small projects to real plants for academics. In fact, this design has generated great interest among the Poppy community as an alternative.

The migration of production technology (from SLS to FDM) has been a success. The reduction of cost and time to produce our platform has been possible thanks to the decision of redesign completely the Poppy project platform. We were able to obtain a robust platform that is easy to produce and assemble. Also it allows future changes like new feet with sensors, or attach accessories to the metal spine.

Finally, the software work done during this project allows to novel users to easily adapt to the biped platform and start working rapidly. The use of the recommended file structures will help to future users to develop their ideas.

Acknowledgement

The results of a project depends more on the people around the author that the author himself, providing important tips and lending a hand when needed. These small acts helps one offering the best, and must be considered in this thesis.

Firstly, thanks to Dimitris Zervas for all the collaboration during all the duration of both master thesis. A friend and workmate that allows to debate ideas to find the best solution. Thanks to his work, I could accomplish mine.

Much obliged to both advisors, Dr. Cecilio Angulo and Dr. Manel Velasco. Both believe in us from the beginning, providing all the necessary to develop our projects, when the viability had not been proven yet. Things like that made one continuing working even in the most desperate days, making one not give up until the work is done. Also thanks to Dr. Angulo for all the help provided during the last days.

Last but not least important, the family and friends. Whom their support has become a robust structure where develop this project and accomplish it.

To all of you, thanks for being with this master thesis each moment during the long path.

Bibliography

[1] Matthieu Lapeyre, *Poppy: open-source, 3D printed and fully-modular robotic platform for science, art and education.* Universite de Bordeaux, 2014. English.

[2] Dimitri Zervas, *Implementation of a robot platform to study bipedal walking.* Universitat Politecnica de Catalunya, 2016. English.

[3] Martinez, A. y Fernandez, E., *Learning ROS for Robotics Programming.* PACKT Publishing, 2013

[4] Mark E. Rosheim, *Robot Evolution: The Development of Anthrobotics.* ISBN: 978-0-471-02622-8, November 1994.

[5] Kato, I., Ohteru, S., Kobayashi, H., Shirai, K. & Uchiyama, *Information-power machine with senses and limbs.* In Proc. CISM-IFToMM Symp. on Theory and Practice of Robots and Manipulators, Udine, Italy, 1973

[6] Miomir Vukobratovic, *Contribution to the Synthesis of Biped Gait.* Jan 1972.

[7] A. Takanishi, G. Naito, M. Ishida, I. Kato, *Realization of Plane Walking by the Biped Walking Robot WL-10R.* Department of Mechanical Engineering, Waseda University, 1985.

[8] MCGHEE, R. B., *et al. An approach to computer coordination of motion for energy-efficient walking machines..* Bull. of the Mech. Engineering Lab., 1986.

[9] GARAGE, Willow., *Xml robot description format (urdf).* http://www. ros. org/wiki/urdf/XML, accessed March, 2012, vol. 10.

[10] Pal Robotics., *Packages for running REEM-C in the Gazebo simulator.* https://github.com/pal-robotics/reemc_simulation, accessed June, 2016

[11] DENG, Yelin, *The impact of manufacturing parameters on submicron particle emissions from a desktop 3D printer in the perspective of emission reduction.* Building and Environment, 2016.